CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

# CE141: COMPUTER CONCEPTS & PROGRAMMING

## Chapter – 11

# Pointers

DEPSTAR
Devang Patel Institute of Advance Technology and Research

# Objectives

- To be able to know the concepts of Pointers
- To be able to know how pointer variables are used in a program
- To be able to explain the chain of pointers
- To be able to know the Pointer Expression
- To be able to describe Pointers and Arrays

# Introduction

- In this chapter, we will discuss
  - Understanding Pointers
  - Underlying Concepts of Pointers
  - Accessing the concepts of a variable
  - Declaring Pointer Variable
  - Initialization of Pointer Variable
  - Accessing a Variable through its Pointer
  - Chain of Pointers
  - Pointer Expression
  - Pointer increments and scale factor
  - Pointers and character string
  - Array of  Pointers
  - Pointers as Function Arguments
  - Function Returning Pointers
  - Pointers to Function
  - Pointers and Structures
  - Trouble with Pointers

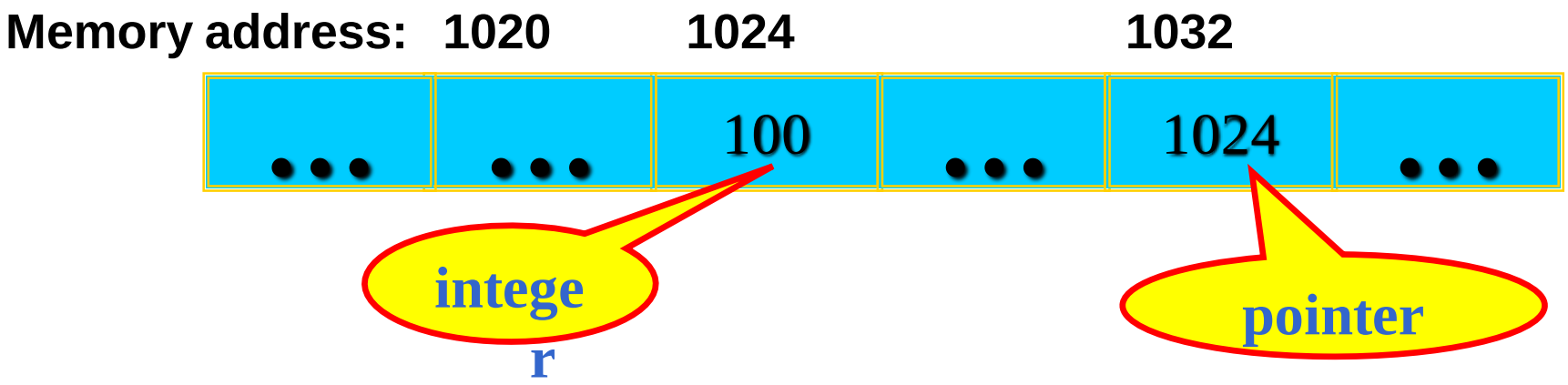
# Understanding Pointers

- The computer memory is the sequential collection of storage cells
- Each cell known as byte, has a number called address associated with it.
- Address starts from zero. Last address depends on memory size.
- Computer having 64K memory will have its last address 65,535.
- At the declaration time systems allocates, some appropriate location to hold the value of the variable

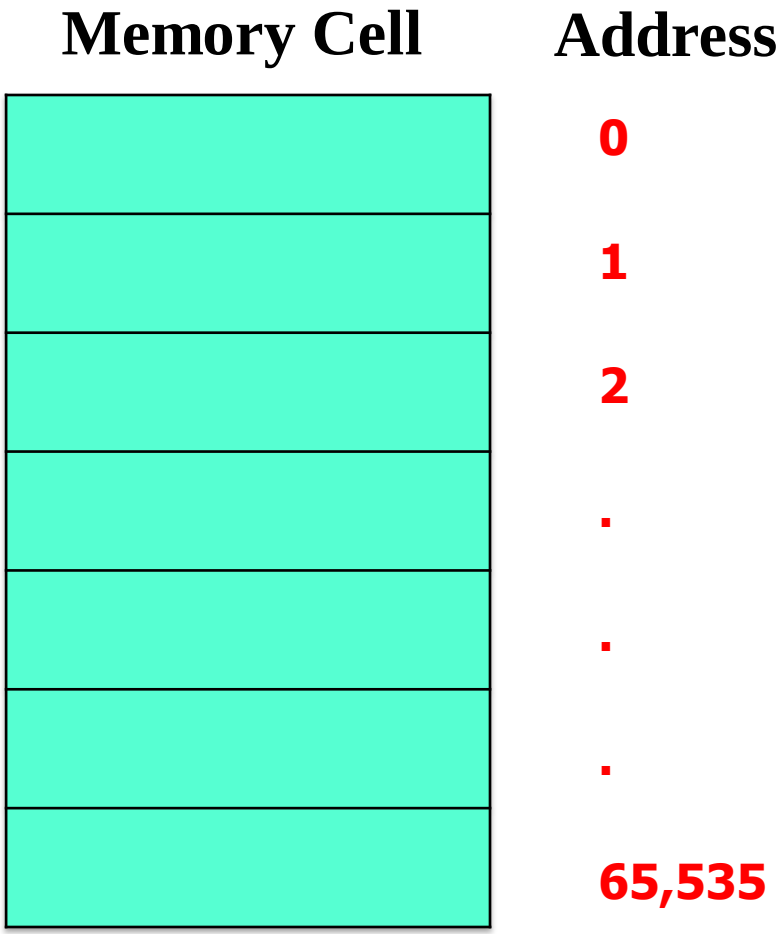
**Variables that hold memory addresses are called pointer variables.**

# Introduction

- Pointers are variables that contain *memory addresses* as their values.
- A variable name *directly* references a value.
- A pointer *indirectly* references a value. Referencing a value through a pointer is called *indirection*.
- A pointer variable must be declared before it can be used.



# Introduction




| Address | Memory | |
|---|---|---|
| . | . | |
| 1000 | 7 | x |
| 1001 | | |
| 1002 | 1000 | y |
| 1003 | | |
| 1004 | | |
| 1005 | | |
| . | . | |

**Variable:** x
**Value:** 7
**Address:** *1000*

# Underlying Concepts of Pointers

Pointers

- Pointer Constants
- Pointer Values
- Pointer Variables

**Pointer Constants**

- **Memory address within the computer**
- **We can not change them**
- **We can only use them to store data values**

**Pointer Values**

- **Can't save the value of memory address directly**
- **Can only obtain the value through the variable stored there using the address operator (&)**

**Pointer Variables**

- **Once we have a pointer value, it can be stored into another variable**
- **The variable that contains a pointer value is called as pointer variable.**

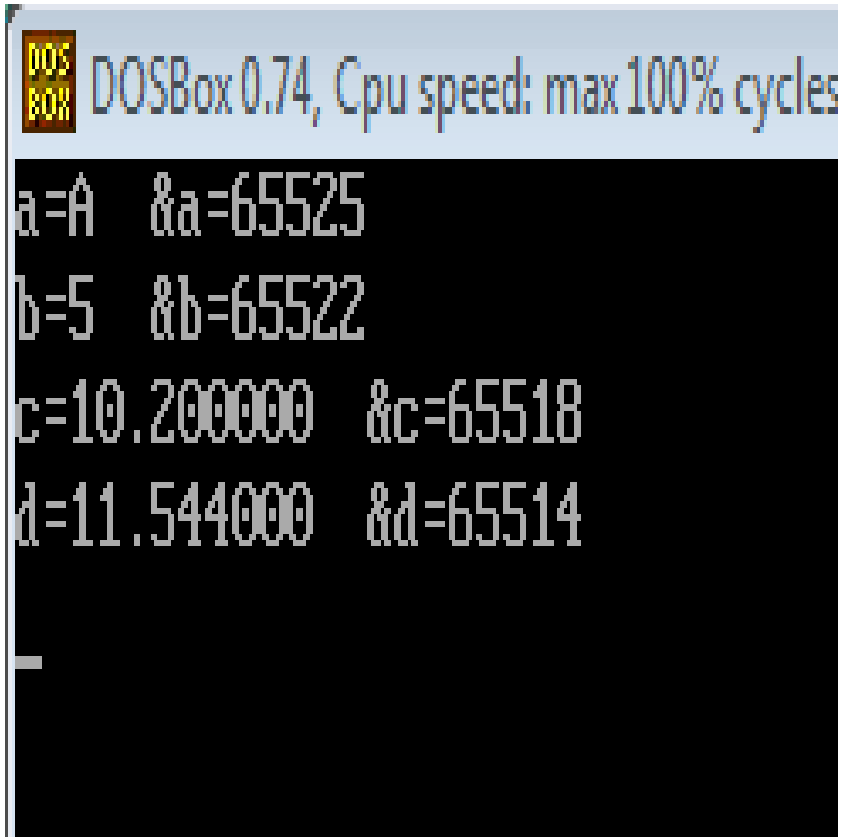
# Accessing the concepts of a variable

- Actual location of a variable is system dependent
- & immediately proceeding a variable returns the address of the variable. For example, p=&a.
- & operator can be used with simple variable or array element.
- Following are illegal use:
  - &125 (Pointing at constants)
  - int x[10];

    &x (Pointing at array names)
  - &(x+y) (Pointing at expression)
  - If x is an array, then expression such as &x[0] and x[i+3]

# Accessing the concepts of a variable

```
#include<stdio.h>
#include<conio.h>
void main()
        {
        char a='A';
        int b=5;
        float c=10.2, d=11.544;
        clrscr();

        printf("a=%c  &a=%u\n",a,&a);
        printf("b=%d  &b=%u\n",b,&b);
        printf("c=%f  &c=%u\n",c,&c);
        printf("d=%f  &d=%u\n",d,&d);

        getch();
        }
```



# Declaring Pointer Variable

- In c, variable must be declared.

  **data_type *pt_name;**

  This tells the compiler three things about the variable pt_name.

  - * tells that pt_name is pointer variable
  - pt_name needs memory location.
  - pt_name points to a variable of type data-type.

  For example, int *a; /* integer pointer */

  - Declares the variable a as a pointer variable that points to an integer datatype.

**The type int refers to the data type of the variable being pointed to by a and not the type of the value of the pointer**

# Declaring Pointer Variable- Pointer declaration style

- Pointers variables are declared similarly as normal variables except for the addition of the unary operator.

**int* p;**

**int *p;**

**Style-1**

```
int *p, x,*q;
```

**Style-2**

```
int x, *p, y;
x=10;
p=&x;
y=*p; /*Accessing x through p */
*p=20; /*Accessing 20 to x */
```

# Initialization of Pointer Variable

- Process of assigning the address of a variable to a pointer variable is known as initialization
- Once pointer variable has been declared we can use the assignment operator to initialize the variable

```
int quantity;
int *p;  /*declaration*/
p=&quantity;  /*initialization*/
```

**or**

```
int *p=&quantity
```

# Initialization of Pointer Variable

- Pointer variables always point to the corresponding type of data
- For example,

```
float a,b;
int x,*p;
p=&a;    /*Wrong*/
b=*p;
```

Will result in erroneous output because we are trying to assign the address of a float variable to an integer pointer

# Initialization of Pointer Variable

- We could define a pointer variable with an initial value of NULL or 0.

```
int *p=NULL;
int *p=0;
```

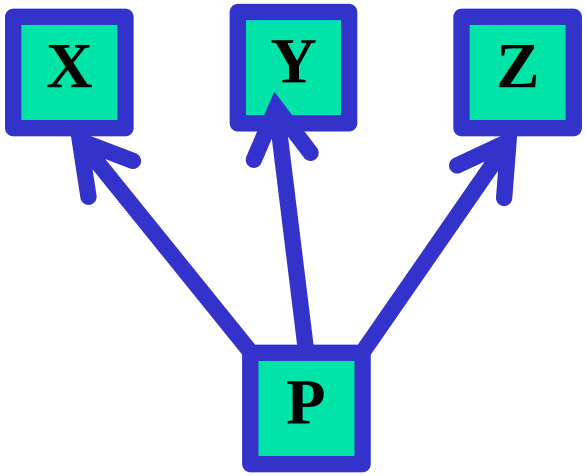
- With the exception of NULL and 0, no other constant value can be assigned to a pointer variable.
- For example, following is wrong:

```
int *p=5360; /*absolute address */
```

# Initialization of Pointer Variable- Pointer Flexibility

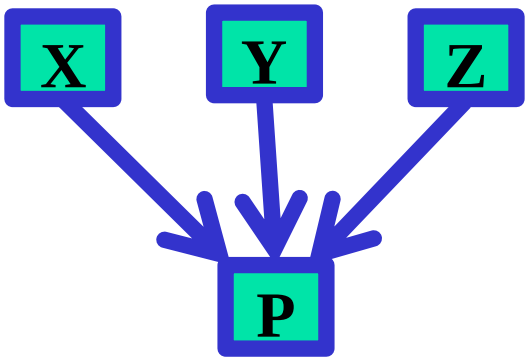
- Same Pointer to different data variables in different statement.

```
int x,y,z,*p;
.............
p=&x;
.............
p=&x;
.............
p=&z;
.............
```




- Different Pointers to point to the same data variable.

```
int x;
int *p1=&x;
int *p2=&x;
int *p3=&x;
...................
...................
```

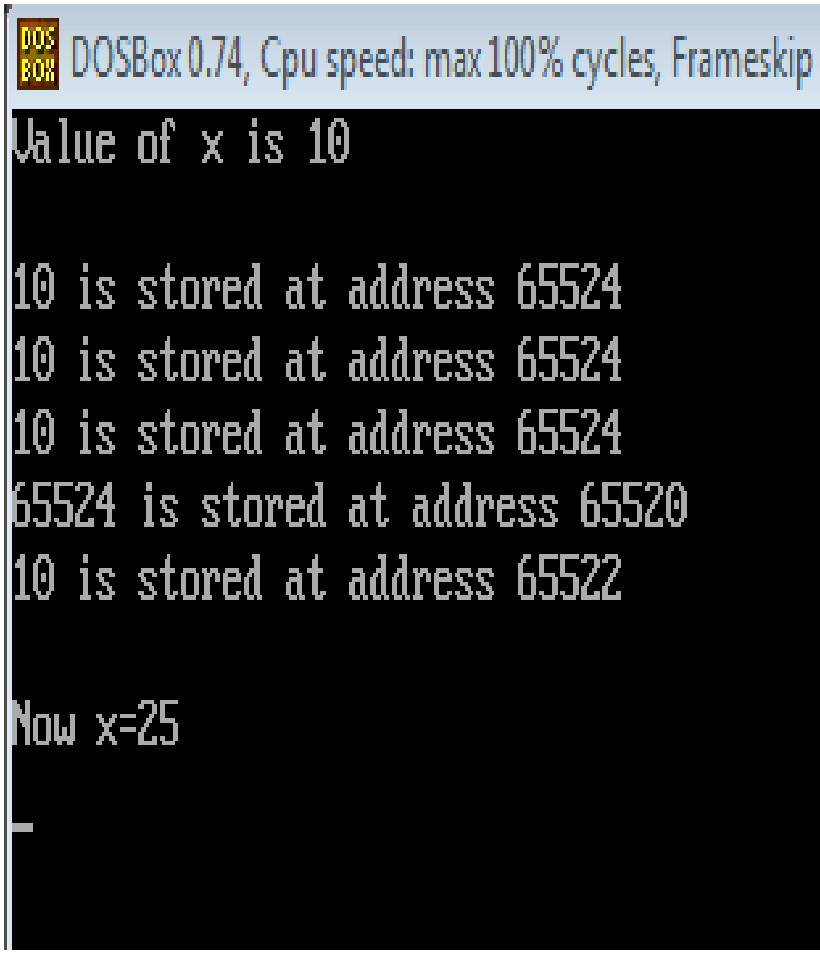


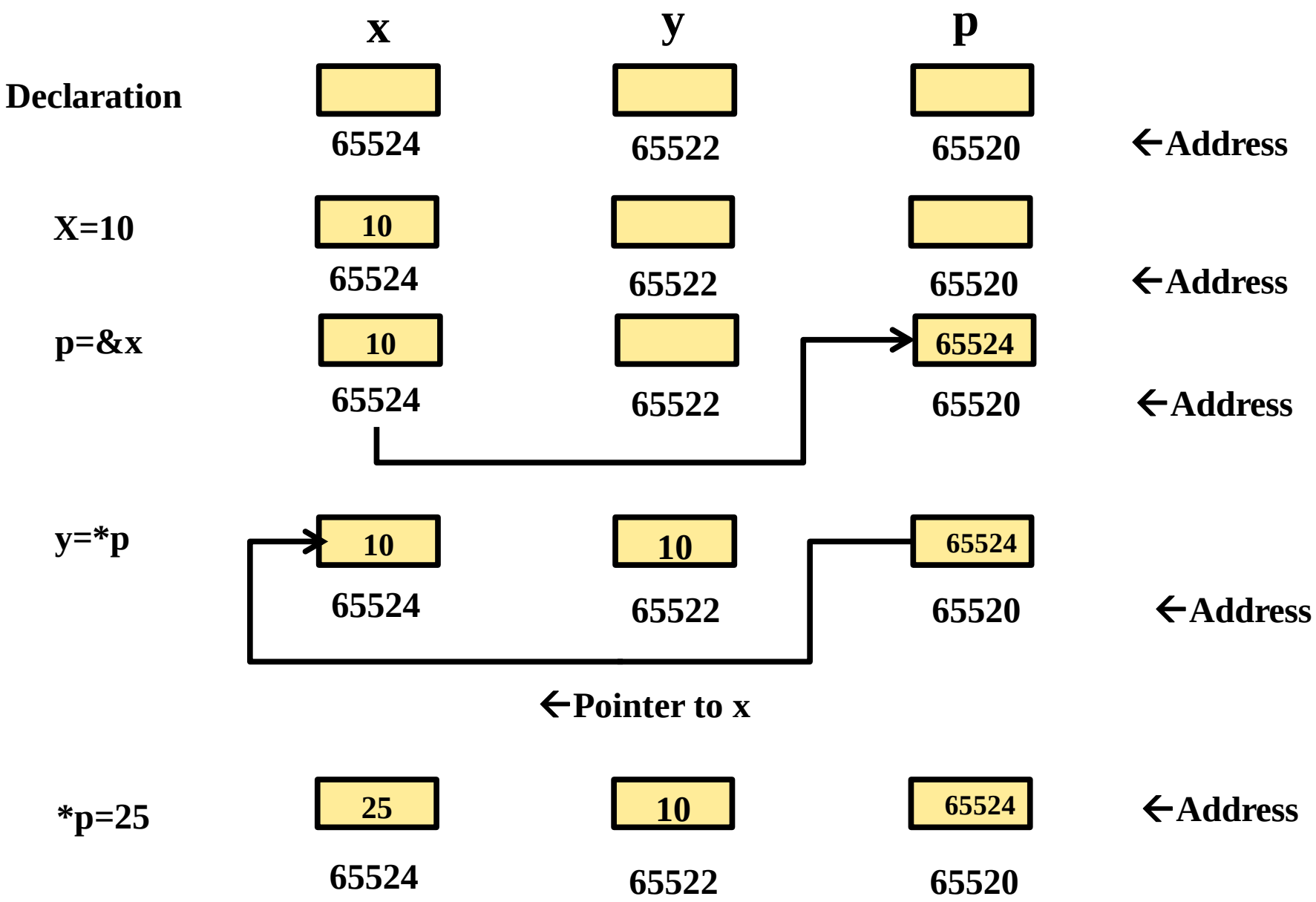
# Accessing a variable through its Pointer

- How to access the value of a variable using pointer?

```
int q,*p, n; //Declaration of q, n and pointer p
q=10; //Initialization of q
p=&q // Initialization of the address of the p
n=*p; //returns the value of a variable of which address is stored
```

```
void main()
{
int x,y;
int *p;
x=10;
p=&x;
y=*p;
clrscr();
printf("Value of x is %d\n\n", x);
printf("%d is stored at address %u\n",x, &x);
printf("%d is stored at address %u\n", *&x, &x);
printf("%d is stored at address %u\n", *p, p);
printf("%u is stored at address %u\n", p, &p);
printf("%d is stored at address %u\n", y, &y);
*p=25;
printf("\nNow x=%d\n", x);
getch();
}
```

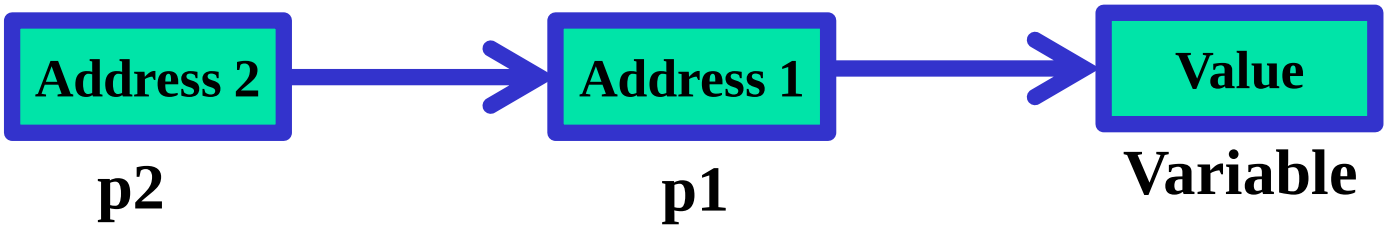





# Illustration of Pointer Assignments

# Chain of Pointers

- Possible to make a pointer to another pointer, thus creating a chain of pointer as shown




- Pointer p2 contains the address of pointer variable p1, which points to the location that contains the desired value
- Declaration of Pointer to Pointer: int **p2

```
void main()
        {
        int x,y;
        int *p,**z;
        x=10;
        p=&x;
        y=*p;
        z=&p;
        clrscr();
        printf("Value of x is %d\n\n",x);
        printf("x=%d is stored at address &x=%u\n",x,&x);
        printf("*&x=%d is stored at address &x=%u\n",*&x,&x);
        printf("*p=%d is stored at address p=%u\n",*p,p);
        printf("p=%u is stored at address &p=%u\n",p,&p);
        printf("y=%d is stored at address &y=%u\n",y,&y);
        *p=25;
        printf("\nNow x=%d\n",x);
        printf("z=%u is stored at address &z=%u\n **z=%u",z,&z,**z);
        getch();
        }
```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Pro

```
Value of x is 10

x=10 is stored at address &x=65524
*&x=10 is stored at address &x=65524
*p=10 is stored at address p=65524
p=65524 is stored at address &p=65520
y=10 is stored at address &y=65522

Now x=25
z=65520 is stored at address &z=65518
 **z=25_
```

# Pointer Expressions

- Pointer variables can be used in expressions
- For example, p1 and p2 are declared and initialized pointer variables, then

**Y=*p1+*p2**

**X=(*p1)*(*p2)**

p1*p2\\invalid

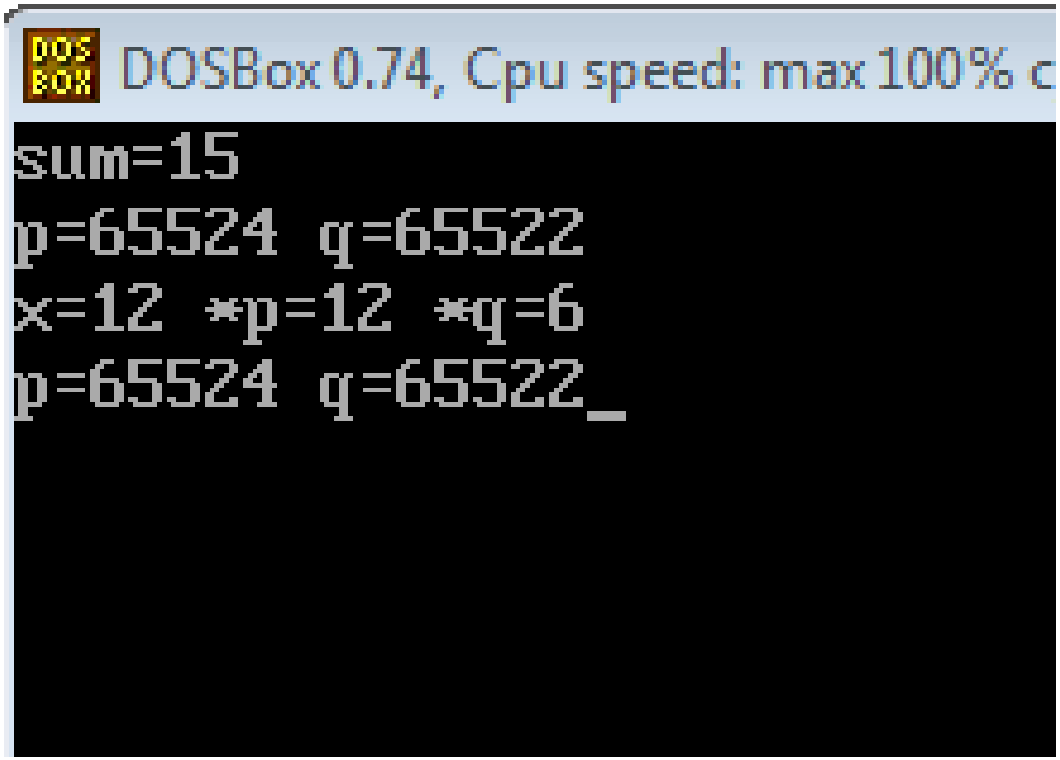
# Pointer Increments and Scale Factor

- Pointers can be incremented like

  **p1=p1+1 or p1++**

- If p1 is an int pointer with initial value , say 2800,then after the operation p1=p1+1, the value of p1 will 2802, not 2801

```
void main()
        {
        int x=10,y=5,sum,sum1;
        int *p,*q;

        p=&x;
        q=&y;
        clrscr();
        sum=*p+*q;
        printf("sum=%d",sum);
        printf("\np=%u q=%u",p,q);
        //invalid pointer addition        sum1=p+q;
        *p=*p+2;
        ++*q;
        printf("\nx=%d *p=%d *q=%d\np=%u q=%u",x,*p,*q,p,q);
        getch();
        }
```



# Pointers and Arrays

- When an array is declared, the compiler allocates a base address and sufficient amount storage to contain all the array in contiguous memory location.
- The base address is the location of the first element (index 0) of the array

| X[0] | X[1] | X[2] | X[3] | X[4] |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1000 | 1002 | 1004 | 1006 | 1008 |

```
void main()
  {
  int *p,sum=0,i=0;
  int x[5]={1,2,3,4,5};
  p=x;
  clrscr();
  printf("\tElement\tvalue\tAddress\n");
  while(i<5)
          {
          printf("\tx[%d]\t%d\t%u\n",i,*p,p);
          sum=sum+*p;
          i++,p++;
          }
  printf("sum=%d\n",sum);
  printf("&x[0]=%u\n",&x[0]);
  printf("p=%u\n",p) ;
  getch();
  }
```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip (

```
        Element value   Address
        x[0]    1       65514
        x[1]    2       65516
        x[2]    3       65518
        x[3]    4       65520
        x[4]    5       65522
sum=15
&x[0]=65514
p=65524
```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, P

```
        Element value   Address
        x[0]    1       65514
        x[1]    1       65514
        x[2]    1       65514
        x[3]    1       65514
        x[4]    1       65514
sum=5
&x[0]=65514
p=65514
```

**If we remove p++ from the while loop then following output will be generated**

# Pointers and Arrays

- We can access every value of x using p++ to move one element to another.
- The relationship between p and x is shown as:

  - **P=&x[0] (=65514)**
  - **P+1=&x[1] (=65516)**
  - **P+2=&x[2] (=65518)**
  - **P+3=&x[3] (=65520)**
  - **P+4=&x[4] (=65522)**

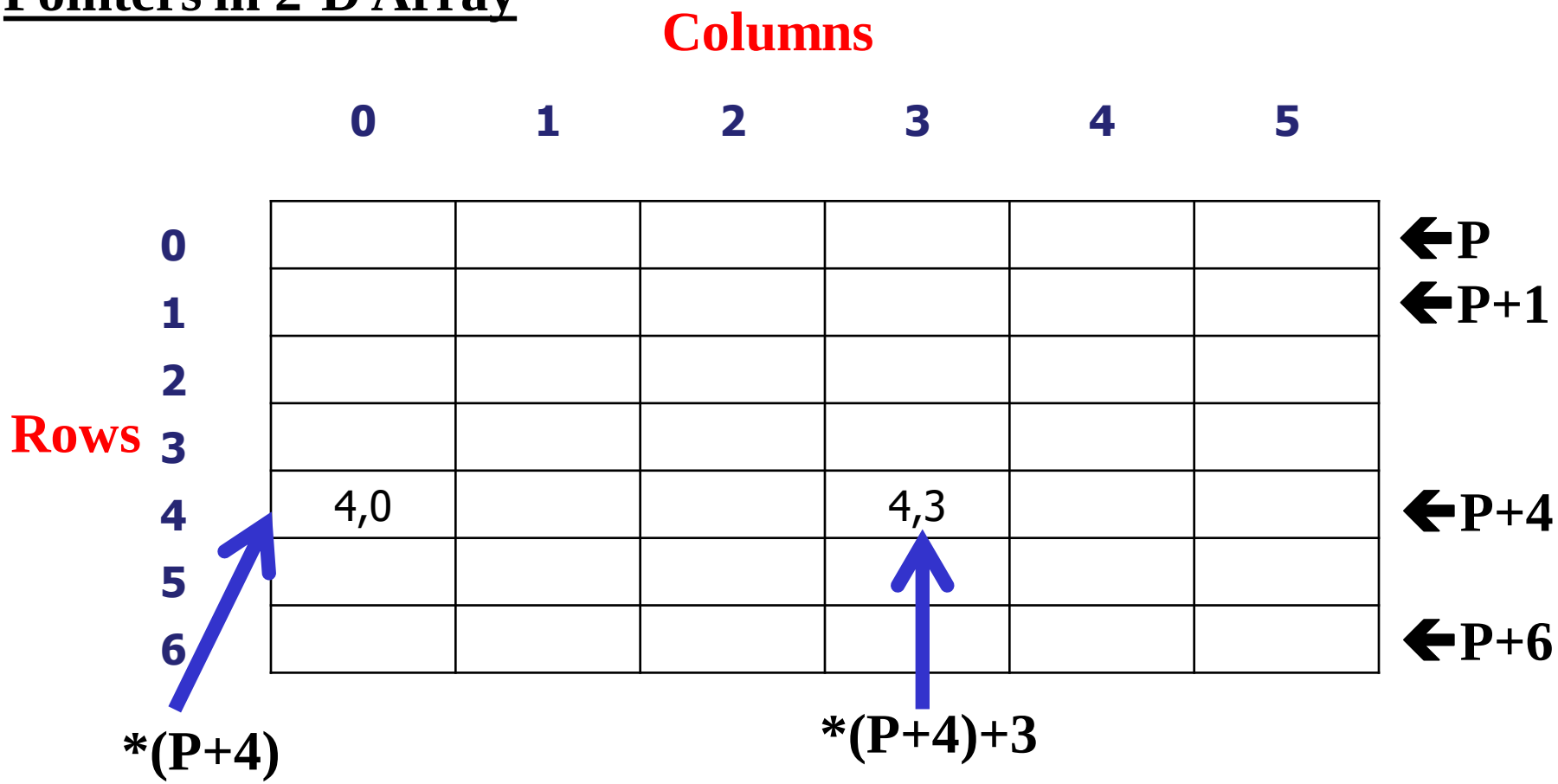
- Address of element is calculated as:

Address of x[3]=65514+(3*2)=65520

# Pointers and Arrays

- Pointers can be used to manipulate two-D arrays as well
- In one-D array x, the expression
  - ***(x+i) or *(p+i)**
- Two-D array can be represented by
  - ***(*(a+i)+j) or *(*(p+i)j)**

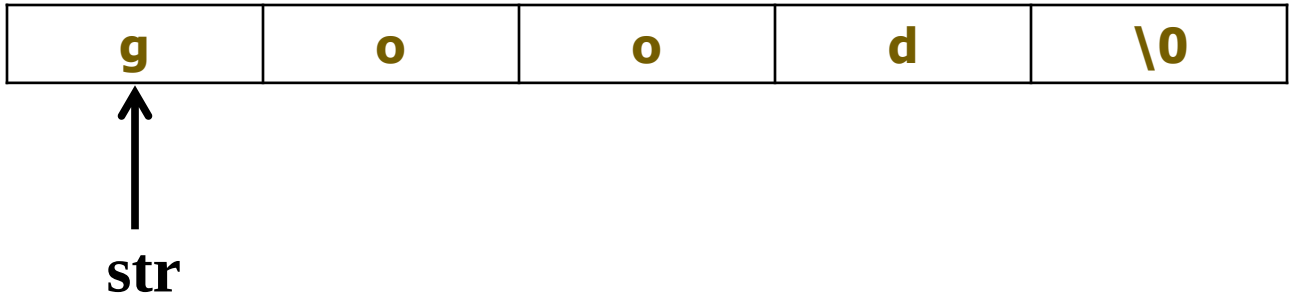
# Pointers in 2-D Array




| | |
|---|---|
| P | Pointer to the First Row |
| p+i | Pointer to the ith Row |
| *(p+i) | Pointer to first element in the ith Row |
| *(p+i)+j | Pointer to jth element in the ith Row |
| *(*(p+i)+j) | Value stored in the cell (i,j) |

# Pointers and Character strings

- C provides alternative method to create strings by using pointer variables of type char.
- **char *str="good"**



# Pointers and Character strings

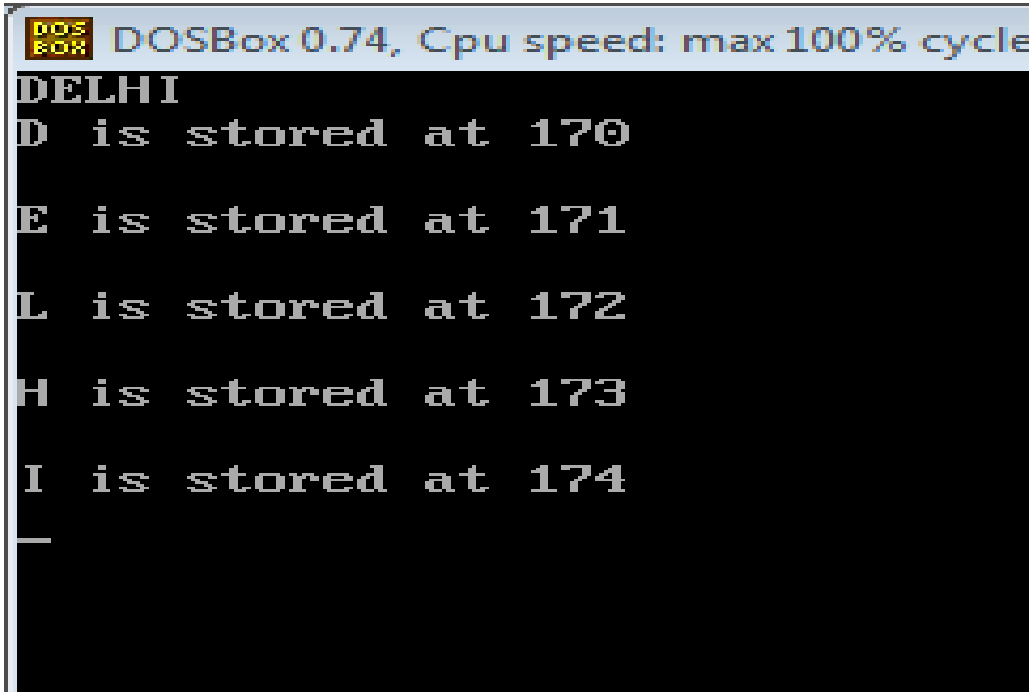
- To print the content of the string1

```
char *string1;
pritntf("%s", string1); or
puts(string1);
```

- Although string1 is a pointer to the string, it is also the name of the string.
- There for we do not need to use indirect operator *.

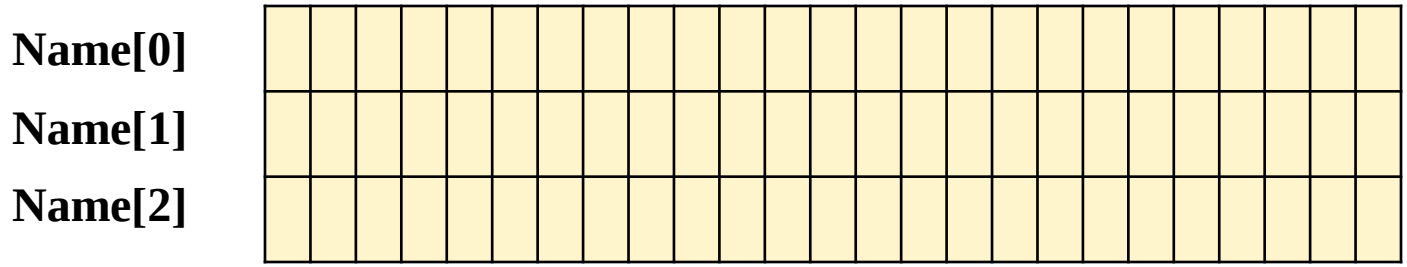
# Pointers and Character strings

```
void main()
        {
        char *name="DELHI";
        char *p=name;

        clrscr();
        printf("%s",name);

        while(*p!='\0')
                {
                 printf("\n%c is stored at %u\n",*p,p);
                 p++;
                }
        getch();
        }
```

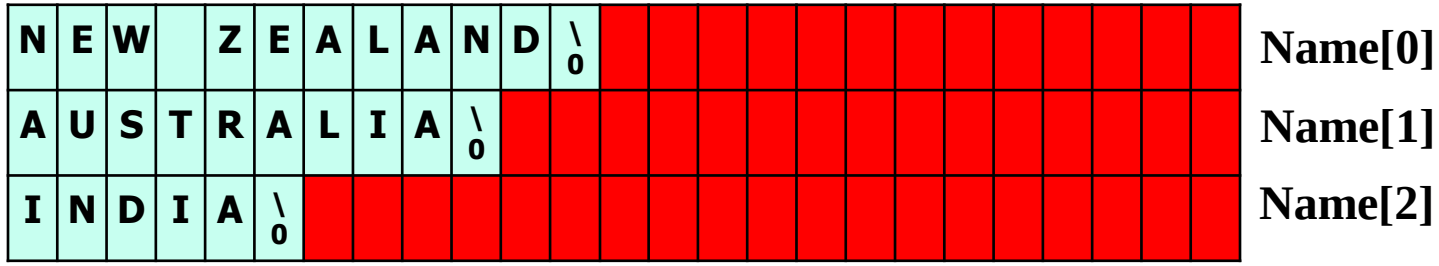


# Array of Pointers

- **Important use of pointer:** To handle table of String
- char name [3][25];
- Following is the table named "name" which contains 3 names, each with a maximum length of 25 characters (including null character).




```
char *name[3]={
  "New Zealand",
   "Australia",
   "India"
 }
```



# Array of Pointers

- The following statement would print out all three names

```
for(i=0;i<=2;i++)
    printf("%s \n", name[i]);
```

- To access jth character in the ith name, we may write as

```
*(name[i]+j)
```

- The character arrays with the rows of varying length called **"Ragged Arrays"**

# Array of Pointers

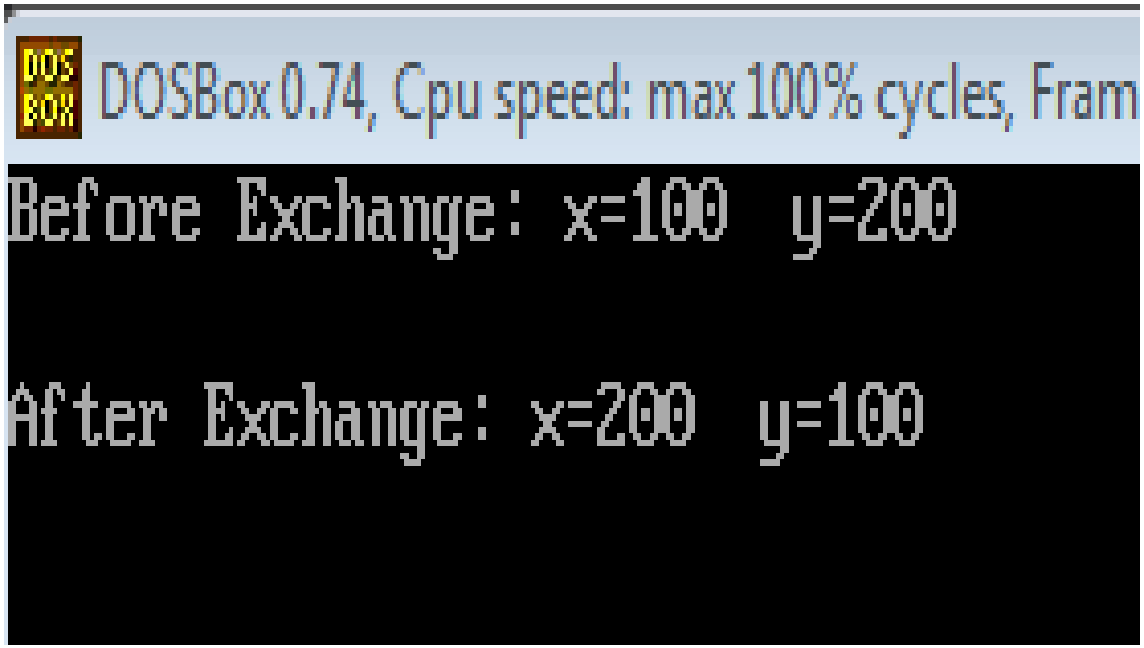
## Difference between *p[3] and (*p)[3]

- Since * has a lower precedence than [], *p[3] declares p as an array of 3 pointers
- (*p)[3] declares p as a pointer to an array of 3 elements

# Pointers as Function Arguments

- The address should pass to the function as an argument
- The parameters receiving the address should be pointers
- The process of calling a function using pointers to pass the address of variables is known as call by reference.
- The process of passing actual value of variables is known as call by value.

```
int swap(int*,int*);
void main()
        {
        int x,y;
        x=100;
        y=200;
        printf("Before Exchange: x=%d  y=%d\n\n",x,y);
        swap(&x,&y);
        printf("After Exchange: x=%d  y=%d\n\n",x,y);
        getch();
        }

int swap (int *a,int *b)
        {
         int t;
         t=*a;
         *a=*b;
         *b=t;
        }
```



# Pointers as Function Arguments

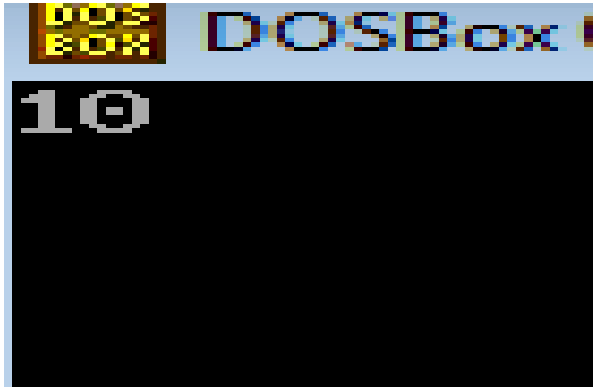
**Note:**

- The function parameters are declared as pointers
- The dereferenced pointers are use in the function body
- When the function is called, the addresses are passed as actual arguments

# Function Returning Pointers

- Function can return multiple values through pointer parameters

```
*large();
void main()
        {
        int x=10,y=5,*p;
        clrscr();
        p=large(&x,&y);
        printf("%d",*p);

        getch();
        }
*large(int *a,int *b)
        {
        if(*a>*b)
                return(a);
        else
                return(b);
        }
```



# Pointer to Function

- A function, like a variable, has a type and address location in the memory.
- It is therefore, possible to declare a pointer to a function
- We can make a function pointer to point a specific function by simply assigning the name of the of the function to the pointer
- For example

```
double mul (int , int);
double (*p1)();
p1=mul;
```

- To call mul, we may now use the pointer p1 with the list of parameters

**(*p1)(x,y);/*Function call*/** is equivalent to

# Compatibility and Casting

```
int x;
char *p;
p=(char *) &x;
```


**When the variable datatype and pointer variable data type is different then casting is necessary**

# Compatibility and Casting

**Void pointer:**

- It is generic pointer
- All the types can be assigned to void pointer without casting.
- For example

```
int x;
char y;
void *v;
v=&x;
v=&y;
```

# Pointers and Structures

```
struct inventory
  {
    char name[30];
    int number;
    float price;
    }product[2], ptr*;
```

product: Data object, array of two elements, each of the type strut inventory

ptr: pointer to data object of struct inventory

**ptr=product ;\\assignment**

**Would assign the address of 0th element of product to ptr.**

# Pointers and Structures

- The pointer ptr will now point to product[0].
- Members can accessed by:

ptr->name

ptr->number

ptr->price

Or (*ptr).number

**→Selection Operator**

```
for(ptr=product; ptr<product+2; p++)
        printf("%s %d %f \n",ptr->name, ptr->number, ptr->price);
```

# Troubles with pointers

- Assigning values to uninitialized pointers

```
int *p, m=100;
*p=m; /*Error*/
```

- Assigning value to a pointer variable

```
int *p,m=100;
p=m; /*Error*/
```

- Not dereferencing a pointer when required

```
int *p,x=100;
p=&x;
printf("%d",p); /*Error*/
```

- Assigning the address of an uninitialized variable

```
int m,*p;
p=&m;
```

# Previous year questions

- Differentiate: Call by value and Call by reference [5 marks]
- Write a program which adds two integer numbers using pointers only [3 marks]
- State true/false: Pointers reduce the length and complexity of a program [1 mark]
- What do you mean by pointer? Explain advantage and disadvantage of pointer?

# Previous year questions

- **Write the output of the code: [1 marks]**

```
#include<stdio.h>
void main(){
int i=3;
int *j;
int **k;
j=&i;
k=&j;
printf("%d",*(&i));
printf("%d",**k);
}
```

# Previous year questions

- Check following C code. Find runtime error if any. Also discuss advantages of pointer to function. [4 marks]

```
int *function(){
int x=10;
return(&x);
}
void main(){
printf("%d",*(function()));
}
```

# Previous year questions

- Define pointer. How we can access any variable through its pointer?[2 marks]
- Write a program using pointers to compute the sum of all elements stored in an array of size 5. [4 marks]
- State ture/false: Pointer is a variable that stores the address of the another variable. [1 mark]